RSS Team 5 Challenge Design Outline

Eric Wieser[1,2], Ernie Ho[1], Shi-ke Xue[1], Steven Homberg[1], Winter Guerra[1]

Abstract— **The project is a culmination of the work and learning this semester about the provided racecar and ROS. The racecar must autonomously navigate through MIT's basement tunnels and complete an obstacle course in the fastest time possible. This is a race against other teams in two parts - the robot is timed running the course on its own and also timed while concurrently running the course with other teams' racecars.**

I. Problem statement

The goal of the challenge is to enable a mobile robot platform to efficiently traverse a static environment with known map containing several unknown obstacles and targets.

The course the robot must traverse is in a portion of the MIT tunnels below MIT, with flat ground and walls. This provides a reasonably uniform environment for the robot to traverse. Beyond the baseline of the tunnel, there are several additional obstacles presented by the environment. People will be standing at the side in various portions of the course, which can alter the observed environment compared to that which is expected based on the map. There are also cone-like obstacles in the middle of the course in several places which the robot must avoid.

Positive Goals

Reaching the end of the course through the tunnels as fast as possible is the ultimate goal of the challenge. In addition to completion of the course, there are colored targets on the ground which, if touched by the robot, will increase the robot's score.

Negative Goals

Colliding with one of the cone-like obstacles placed on the course results in a penalty to the robot's score.

Although not explicitly penalized, other crashes, for example with the walls, decrease the robot's score by postponing the completion of the goal.

II. Assumptions

- We have "instantaneous" control of the car's angular velocity using the steering. This simplifies the RRT motion update steps and simplifies the generation of valid paths without greatly adversely affecting performance.
- We assume that every path generated by the RRT is traversable regardless of the speed at which we are traveling. This helps simplify our path generation as the RRT can ignore speed as a variable. This is accomplished by constraining our motion update to the lower bound of our steering capabilities at speed.
- We assume from previous experience that our ZED camera will give us data that is outdated by half a second or more and also has a slow update time of 5HZ or so. Therefore, we assume that we will have to use tf's time travel features to allow our sensor processing suite to work correctly.

III. Technical approach

A. Vision (Ernie)

Using camera to recognize the cone and markers in front of the robot. It is optional to recognize people and other robots as well. Basically we use lab4 approach to detect the cone. We are going to use OpenCV as a tool to recognize the obstacles based on their features (color and shapes). Alternatively, we can use NVIDIA visionworks to accelerate the obstacle recognition or learning approach to recognize obstacles.

B. Localization and Mapping (Steven)

In order to understand the environment, the robot must both know where it is in the environment, and the occupancy of the relevant parts of the environment. The laser scanner data in conjunction with the odometry estimates computed by the motor controller should provide the required information to infer these.

In the environment of the challenge, most of the surroundings is known prior to the run, while there are some obstacles which are not known until detected online. Thus, the task of understanding the environment contains some elements of simple localization, and some elements of mapping. This means that a localization algorithm is not sufficient for the task, while a SLAM algorithm is unnecessarily powerful.

Our decision between an approach involving localization augmented with mapping and an approach applying SLAM will be motivated by performance. We will test whether Monte-Carlo localization, as implemented in the AMCL package, is able to effectively localize the robot in the tunnel in the presence of some obstacles not present in the map, including cones or similar obstacles in the middle of the path as well as people standing on the sides, possibly blocking off significant map features like side-corridors. If it does localize the robot well, then mapping the obstacles relative to the well-localized robot is not computationally expensive. We will test alternatives for the method of conducting this mapping, including full ray-tracing or the more simplistic method of treating cells as occupied only if one of the laser scan ranges lies in that cell.

[1]Massachusetts Institute of Technology, EECS
[2]University of Cambridge

Alternatively, if AMCL is not able to localize the robot as computationally efficiently or as robustly as using the hector-slam package online, then simply using that will serve to both localize the robot and detect previously unexpected obstacles.

Finally, the robot will ideally be able to determine the type of obstacle it is facing for planning purposes. If faced with a complicated maze of cones, it may be the case that it is beneficial to our score to plow through a cone and save time rather than spending time planning the way through the maze. Cones can be treated like this, but other obstacles like walls, people, and other robots cannot, so making these planning decisions requires a more involved type of obstacle detection. If we find that it is reasonably easy to do this, either based on the size or shape of the perceived obstacle in the laser scan or based on other sensor input like the camera, then we will do so.

C. Low-level Path planning (Eric)

In lab6, we used an RRT that found statically achievable paths between points - that is, paths which follow arcs that an ackermann car is able to drive on. Unfortunately, this RRT was slow to converge, and as is the nature of RRTs, did not always produce optimal paths. Since the final project is a race, optimality is of interest to us.

To improve convergence, we need to adjust our distance metric and sampling routine, and profile our code. It's entirely possible that our choice of these two functions has resulted in a degenerate RRT algorithm which is no longer probabilistically complete. Analyzing this is probably out of scope for this class, but reading some papers to see how else it's been done could provide insight. Profiling has already been effective at locating bottlenecks, but there's still efficiency to be gained. We should consider switching to using C++ or at least Cython to provide speedups as well.

To improve optimality, we should consider implementing RRT*. Care must be taken to use the right version of optimality - distance- and time-optimality are not the same thing when dynamics are taken into account. Again, the race factor makes time-optimality desirable, so the RRT implementation needs to understand the robot dynamics. We'll assume that modeling forwards velocity is enough, and stick with our assumption that the steering assembly turns instantaneously.

D. High-level Path planning (Shi-Ke)

At a higher level, path selection is determined not just by distance but by the speed it takes to traverse the path. Although cones and markers each penalize or reward time, it is not necessarily the correct strategy to always path to avoid cones or to chase markers. A higher level strategy needs to take into consideration the trade-offs for avoiding or approaching the two targets in terms of time.

It's possible that a cone is detected too late to route around without significant time loss, such as if one is spotted right as the racecar turns a corner. Similarly, it's possible for markers to be detected too late to feasibly change route to cross over them. As a result, the high level path planner must weigh the costs of missing a marker or hitting a cone against the cost of decelerating to pass over a marker or avoiding a cone. Of course, this cannot be applied to all obstacles - the car should avoid all other obstacles whenever possible.

E. Path following (Winter)

To optimize the speed at which our car runs the course, we would like our path follower control system to be tightly connected to our higher level RRT* path planner. To do this, it should try its best to follow the exact path the RRT outputs and also output signals describing the robot execution state.

In our previous implementations of our path follower node, we described the goal path using a list of positional waypoints. However, this representation of our path did not take into account the physical dynamic limitations of an Ackermann controlled car and ran into issues of slow convergence to the desired path when waypoints were spaced far apart. To resolve this issue, we are going to upgrade our path follower node to take in a list of arcs that are calculated by the RRT motion update step. We are then going to interpolate this list of arcs to create a completely defined path for the robot to follow. This upgrade in path representation should allow our robot to more faithfully follow detailed paths around complex obstacles without creating much computational overhead.

Lastly, to optimize our computation time effectively, we will also make sure that our path follower node reports back detailed path progress metrics to the RRT* node such that this data can be used to prune old upstream paths and speed up the path searching process. Additionally, the outputted path completion metrics could also be used upstream in the RRT* node for optimizing paths using predictive state lookahead.

IV. Capability milestones

April 20	Baseline functioning robot which completes the course with no obstacles, either by localizing and following a predefined path or following a wall.
April 27	Navigate the course with localization and the RRT path planner in the absence of obstacles; detect markers
May 4	Dry run - Navigate the course with obstacles; RRT* working in simulation
May 9	Navigate the course with obstacles, using high-level planner to pass over markers.

V. Decision making process

The general decision making process for both how to divide work and for our implementation for the final project has been to communicate important decisions during team meetings. General consensus or at least agreement can be reached on major decisions - minor issues can be talked about online through Slack.

Division of labor

In order to keep track of which tasks are assigned to which people, we will use github issues. This allows us to label tasks by the technical topic they pertain to, so we can see which groups of tasks are falling behind and need more people.

The current mapping of tasks to people is:

Eric:

- Investigate other RRT models.
- Improve performance of the existing RRT code.

Ernie:

- Detect the cones, markers.
- (Optional) Distinguish static obstacle (wall, feet) with movable obstacles(robots, cones).

Shi-ke:

- Implementation of RRT* (with Winter).

Steven:

- Optimize AMCL performance on-robot.
- Finish implementing local-grid mapping.
- Find metric to distinguish wall-like obstacles from movable obstacles.

Winter:

- Upgrade path representation in order to assure path completeness.
- Help Shi-Ke implement and improve RRT* node.
- Integrate path following metrics with lookahead and pruning capabilities of RRT* (with Shi-ke).

VI. Self-assessments

Steven

Technical: Over the course of the labs, I gained an appreciation for the computational constraints that arise when implementing things. Thinking about algorithms is something I have a lot of experience with, but figuring out how to speed up constants when actually writing code to execute them was a new experience. I also learned more about managing my code while working in a team.

Communication and Collaboration: The most impactful thing I learned in terms of communication was how to effectively convey information with slides in a presentation. I didn't have much experience with genuinely trying to get people to understand something technical through a presentation, so the tips about formatting slides were quite informative.

Winter

Technical: During 6.141, I learned how to work on complex software architectures as part of a team. Although working as a team to develop software architectures sounds easy, it involves a lot of intercommunication, specifications, and system design something that I did not have a lot of experience in. Additionally, I also learned how to use ROS for the first time.

Communication and Collaboration: Building a robot with a team requires a lot of team communication about system design, tradeoffs, timelines, and feasibility. In 6.141, I learned how to improve my communication such that I could integrate needs from my teammates into my work while also communicating my needs to the team.

Shi-Ke

Technical: The bulk of the technical difficulty has been adapting to new architectures such as ROS. On top of learning new workflows for working with ROS, the algorithms taught in class have been a great experience both to learn and to implement in the labs.

Communication and Collaboration: Having taken 6.UAT, I felt pretty comfortable with individual presentations but I have not had much experience with group presentations, which are very different. 6.141 contributed greatly to my experience doing these types of presentations. Additionally, this was the first group that I had used Slack with, which was a valuable collaboration experience as well.

Ernie

Technical: 6.141 gave me a chance to integrate the knowledge of hardware and software and build the complex system cooperatively which I seldom experienced before. I realize how important to document the code and organize the issues to work as a team.

Communication and Collaboration: I learnt some basic concept of project management before but didn't find a good tool before. In 6.141, we use slack and github as communication tool and I think they are the best combination communication tools which help me keep track not only the conversation but also the progress.

Eric

Technical: 6.141 has introduced me to the ROS infrastructure, and taught me alternate ways of structuring complex systems. It's provided me with great opportunities to work on the implementation of algorithms I've learnt about in previous classes. Having to profile and optimize code is a learning experience for me, as it's something I've not had to think about before.

Communication and Collaboration: It's been a while since I've had to do a group presentation, so this has been valuable to me. 6.141 has also acted as a very effective trial run of slack, a tool I previously dismissed as not solving a problem that I had. I consider myself now converted, and will aim to use it in future group projects!